Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

Claim 1 (previously presented): A method for implementing two types of architectures on a chip, comprising:

receiving instructions from a fetch engine,

determining whether each instruction is a macroinstruction or a microinstruction,

if the instruction is a macroinstruction,

sending the macroinstruction to an emulation engine,

decomposing the macroinstruction into one or more microinstructions,

formatting, by a bundler, the microinstructions into bundles as preferred by the native microarchitecture,

dispatching a bundle in parallel to an execution engine via a multiplexer, and

dispatching additional information to the execution engine, wherein the additional information is contained in bits of the bundle otherwise not required for emulation of the macroinstruction, and

if the instruction is microinstruction, dispatching the microinstruction to the execution engine via the multiplexer;

selecting either the microinstruction from the fetch engine or the bundle from the emulation engine, by using the multiplexer, and

dispatching the selected microinstruction/bundle to the execution engine.

Claim 2 (canceled).

Claim 3 (original): The method according to clam 1 wherein the additional information includes control information from the emulation front end that is sent using a memory, floating point, integer ("MFI") template, wherein the MFI template specifies that the bundle includes a memory instruction in a first syllable, a floating point instruction in a second syllable, and an integer instruction in a third syllable.

Claim 4 (original): The method according to clam 1 wherein the additional information includes an immediate from an emulation front end that is sent by using a memory, long-

immediate, integer ("MLI") template that is interpreted by the execution engine differently, depending upon whether the execution engine is operating in native mode or emulation mode.

Claim 5 (currently amended): A method for implementing two types of architectures on a chip, comprising:

receiving instructions from a fetch engine,

determining whether each instruction is a macroinstruction or a microinstruction,

if the instruction is a macroinstruction,

sending the macroinstruction to an emulation engine,

decomposing the macroinstruction into one or more microinstructions,

formatting, by a bundler, the microinstructions into bundles as preferred by the native microarchitecture,

dispatching a bundle in parallel to an execution engine via a multiplexer, and

dispatching additional information to the execution engine, wherein the additional information is contained in bits of the bundle otherwise not required for emulation of the macroinstruction, and

if the instruction is microinstruction, dispatching the microinstruction to the execution engine via the multiplexer;

selecting either the microinstruction from the fetch engine or the bundle from the emulation engine, by using the multiplexer;

dispatching the selected microinstruction/bundle to the execution engine; and ~~The method according to clam 1,~~

wherein the bundler receives at least one sequence of instructions ("XUOPs"), determines how many XUOPs are received, and when more than one XUOP is received, determines whether the XUOPs must be issued in parallel.

Claim 6 (previously presented): A method for implementing two types of architectures on a chip, comprising:

receiving an instruction from a fetch engine,

determining whether the instruction is a macroinstruction or a microinstruction,

if the instruction is a macroinstruction,

sending the macroinstruction to an emulation engine,

decomposing the macroinstruction into one or more microinstructions,

formatting, by a bundler, the microinstructions into bundles as preferred by the native microarchitecture, wherein the bundler receives at least one sequence of instructions

("XUOPs"), determines how many XUOPs are received, and when more than one XUOP is received, determines whether the XUOPs must be issued in parallel,

dispatching one or more bundles, wherein the dispatching one or more bundles dispatches a plurality of the bundles containing XUOPs when a plurality of XUOPs must be issued in parallel, and the dispatching one or more bundles dispatches one or more bundles per the following rules:

when the XUOPs must be issued in parallel, dispatches plurality of the bundles containing the XUOPs to the execution engine in parallel,

when the XUOPs need not be issued in parallel, determines whether a particular problem exists, and

when the problem does not exist, dispatches a plurality of the bundles containing the XUOPs to the execution engine in parallel,

when the problem does exist, dispatches a plurality of the bundles containing one of the XUOPs to the execution engine,

when only one XUOP is received, determines whether the one XUOP must be issued in parallel with another XUOP, and

when the one XUOP must be issued in parallel, dispatches nothing to the execution engine,

when the one XUOP need not be issued in parallel, dispatches the bundle containing the one XUOP to the execution engine,

if the instruction is microinstruction, dispatching the microinstruction to the execution engine, and

dispatching additional information to the execution engine, wherein the additional information is contained in bits of the bundle otherwise not required for emulation of the macroinstruction.

Claim 7 (original): A method for implementing two architectures on a chip, comprising,

decoding a macroinstruction into one or more microinstructions, through the use of an emulation engine,

formatting the microinstructions into bundles, by use of a bundler, as preferred by the native microarchitecture, wherein the bundler

receives at least one sequence of instructions (an "XUOP"),

determines how many of the at least one XUOP are received, and

when more than one XUOP is received, determines whether the XUOPs must be issued in parallel,

dispatching the bundle to an execution engine, and

dispatching additional information to the execution engine, wherein the additional information is contained in bits of the bundle otherwise not required for emulation of the macroinstruction.

Claim 8 (original): The method according to clam 7 wherein the additional information includes an immediate from an emulation front end that is sent by using a memory, long-immediate, integer ("MLI") template that is interpreted by the execution engine differently, depending upon whether the execution engine is operating in native mode or emulation mode.

Claim 9 (original): The method of claim 8, wherein, when the execution engine is operating in native mode, the MLI template specifies that a third syllable of the bundle contains an integer instruction that operates on an immediate located in second and third syllables of the bundle, and, when the execution engine is operating in emulation mode, the MLI template specifies that the third syllable of the bundle contains an integer instruction that operates on an immediate located entirely within the second syllable.

Claim 10 (original): The method according to clam 7 wherein the additional information includes control information from the emulation front end that is sent using a memory, floating point, integer ("MFI") template, wherein the MFI template specifies that the bundle includes a memory instruction in a first syllable, a floating point instruction in a second syllable, and an integer instruction in a third syllable.

Claim 11 (original): The method of claim 7, wherein the emulation engine delivers a pre-decode bit to the execution engine along with the bundle.

Claim 12 (original): The method of claim 7, wherein the step of determining whether the XUOPs must be issued in parallel uses the following rules:

when the XUOPs must be issued in parallel, issues a plurality of the bundles containing the XUOPs to the execution engine in parallel,

when the XUOPs need not be issued in parallel, determines whether a particular problem exists, and

when the problem does not exist, dispatches a plurality of the bundles containing the XUOPs to the execution engine in parallel,

when the problem does exist, dispatches a plurality of the bundles containing one of the XUOPs to the execution engine,

when only one XUOP is received, determines whether the one XUOP must be issued in parallel with another XUOP, and

when the one XUOP must be issued in parallel, dispatches nothing to the execution engine,

when the one XUOP need not be issued in parallel, dispatches the bundle containing the one XUOP to the execution engine.

Claim 13 (original): A method for implementing two architectures on a chip, comprising:

decoding a macroinstruction into one or more microinstructions, through the use of an emulation engine,

converting the one or more microinstructions into a bundle, using a bundler, the bundle having at least one syllable and having a template that specifies a type of data included in the bundle, wherein the emulation engine delivers a pre-decode bit to the execution engine along with the bundle, and wherein the bundler

receives at least one sequence of instructions (an "XUOP"),

determines how many of the at least one XUOP are received, and

when more than one XUOP is received,

determines whether the XUOPs must be issued in parallel, and

when the XUOPs must be issued in parallel, issues a plurality of the bundles containing the XUOPs to the execution engine in parallel,

when the XUOPs need not be issued in parallel, determines whether a particular problem exists, and

when the problem does not exist, dispatches a plurality of the bundles containing the XUOPs to the execution engine in parallel,

when the problem does exist, dispatches a plurality of the bundles containing one of the XUOPs to the execution engine,

when only one XUOP is received, determines whether the one XUOP must be issued in parallel with another XUOP, and

when the one XUOP must be issued in parallel, dispatches nothing to the execution engine,

when the one XUOP need not be issued in parallel, dispatches the bundle containing the one XUOP to the execution engine,

dispatching the bundle to an execution engine together with a pre-decode bit, and

transferring, by the emulation engine, additional information to the execution engine, wherein the additional information includes an immediate from an emulation front end that is sent by using an memory, long-immediate, integer ("MLI") template that is interpreted by the execution engine differently, depending upon whether the execution engine is operating in native mode or emulation mode.

Claim 14 (original): The method of claim 13, wherein the additional information is contained in bits of the bundle otherwise not required for emulation of the macroinstruction.

Claim 15 (original): The method of claim 13, wherein, when the execution engine is operating in native mode, the MLI template specifies that a third syllable of the bundle contains an integer instruction that operates on an immediate located in second and third syllables of the bundle, and, when the execution engine is operating in emulation mode, the MLI template specifies that the third syllable of the bundle contains an integer instruction that operates on an immediate located entirely within the second syllable.

Claim 16 (original): A method for implementing two architectures on a chip, comprising:

decoding a macroinstruction into one or more microinstructions, through the useof an emulation engine,

converting the one or more microinstructions into a bundle, using a bundler, the bundle having at least one syllable and having a template that specifies a type of data included in the bundle, wherein the emulation engine delivers a pre-decode bit to the execution engine along with the bundle, and wherein the bundler

receives at least one sequence of instructions (an "XUOP"),

determines how many of the at least one XUOP are received, and

when more than one XUOP is received,

determines whether the XUOPs must be issued in parallel, and

when the XUOPs must be issued in parallel, issues a plurality of the bundles containing the XUOPs to the execution engine in parallel,

when the XUOPs need not be issued in parallel, determines whether a particular problem exists, and

when the problem does not exist, dispatches a plurality of the bundles containing the XUOPs to the execution engine in parallel,

when the problem does exist, dispatches a plurality of the bundles containing one of the XUOPs to the execution engine,

when only one XUOP is received, determines whether the one XUOP must be issued in parallel with another XUOP, and

when the one XUOP must be issued in parallel, dispatches nothing to the execution engine,

when the one XUOP need not be issued in parallel, dispatches the bundle containing the one XUOP to the execution engine,

dispatching the bundle to an execution engine together with a pre-decode bit, and

transferring, by the emulation engine, additional information to the execution engine, wherein the additional information including control information from the emulation front end that is sent using a memory, floating-point, integer ("MFI") template, wherein the MFI template specifies that the bundle includes a memory instruction in a first syllable, a floating point instruction in a second syllable, and an integer instruction in a third syllable.

Claim 17 (original): The method of claim 16, wherein the additional information is contained in bits of the bundle otherwise not required for emulation of the macroinstruction.